HVDC

Main.c

Initializations – inside Main function

```
void main(void)
{
vMcalInit();
vBswInit();
Rte_vInit(); /* application initialize */
#if(!DISABLE_MPU)
    Mem_CfgAccProt(); // Memory Protections
#endif /*!DISABLE_MPU*/
#if(PMA_ENABLE)
    Pma_vInit(); //Post Mortum abort data- to find reasons of unintended reset if any
#endif/*PMA_ENABLE*/
ExtWdgInit(); //Initialize external WD, HVDC both ext and int WD. LVDC only int wd
 Sys_vEnableDog(); //Enable internal WD
 vMcalStart(); //Here we start the timers pwm etc..
 vInit_Gpio_PWM(); //initialize pwm gpios (muxing)
 while(1)       //infinite while loop
{
vMainTask();
}
```

- Mcal init- Calls init functions of all mcal modules, defined in the respective modules
- Bsw init:

```
dbc_vInit();  //dbc file intergaration for int CAN, dbc file contains purely signal
information like signal length, transmitting node, receiving node etc..
 tp_vInit();     //CAN TP
 uds_vInit();   //diagnostics
 dbc_messageInit();
 comserv_vInitSignals();
/* comserv takes the dbc data to upper layers, comserv initsets messages to default
values, cmoserv tx put analog values to CAN bus*/
#if (Scope_Integration) /* Applicable for Scope integration – directly write to RAM
using CAN*/
 Scope_Init();
#endif /* Applicable for Scope integration */
```

- In mcal init we just initialize, not start because we have other modules to initialize. Once other modules are initialized, we call for vMcalStart which enables clocks , timers and interrupts. Interrupts are started once this function is called

- After vMcalStart we initialize GPIOs PWM muxing

ISRs

- Main.c contains all the ISRs for 40us,1ms and 10ms

Functions for SW info

- Functions are defined for getting SW version, Build date and Build time
- These are mostly hardcoded to some defined macros

Sys.c

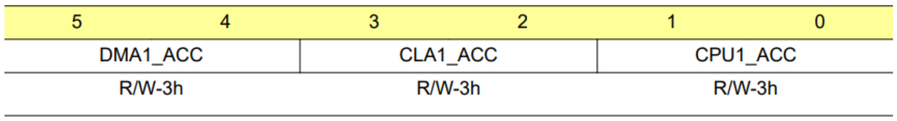
Sys_vInit()

```
void Sys_vInit(void)
{
/*Like PIE, the CPU provides flag and enable flags for CPU interrupts
IER – Interrupt Enable Register: Enables the interrupt
IFR -Interrupt Flag Register*/
/* Disable and clear interrupts: For disabling CPU interrupts*/
        DINT;   //disable interrupts globally
        IER = 0u; //clearing CPU interrupt flag and enable register
        IFR = 0u;
/*This process has to be done before system initialization. We also need to disable WD for
initialization so that it wont get timed out and get reset*/
        vDisableDog();
/*Initialise system clock and PLL*/

        vInitSysPll(XTAL_OSC, IMULT_10, FMULT_0, PLLCLK_BY_2);
        vHoldPeripheralReset(); /* Holds/disable the peripherals we don't use*/
        vInitPeripheralClocks(); /* Initialize all peripheral clocks */
        vSetPeripheralAccess();
/* In hvdc application, we are only using CPU, no CLA or DMA. So we need to set access of
peripherals to CPU only. Here we also disable access to peripherals we don't use */
         vLockConfig();
 /* To lock the configuration so that no one can change the values during run time */
          /* Initialize static variables */
          SysSafety.CLK1 = OK;
          SysSafety.CLK89 = OK;
          SysSafety.SYS4 = OK;
} /* Sys_vInit */
```

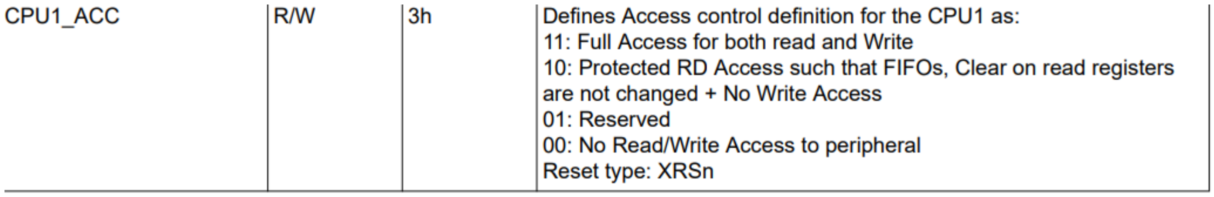
- vInitPeripheralClocks():
  - Enables clock gating to necessary peripherals
  - CpuSysRegs -> PCLKCRx : contains registers for providing CLK access to different peripherals

    - Eg:
      CpuSysRegs.PCLKCR0.bit.DMA = 0; //*Since we don't use DMA, set to 0*
      CpuSysRegs.PCLKCR0.bit.CPUTIMER0 = 1; //*Since we use cpu timer*
    - Like this clock access to all peripherals are defined here
    - Note: ADC loop back test is a test used for checking ADCs working. Here value sent by ADC is read back using a DAC and is cross checked. We use DACA for ADC loop back test. So CLK for DAC is enabled for ADC2 Loopback test, It will be disabled after the test
- vHoldPeripheralReset:
    - Resets the peripheral that are not used
    - DevCfgRegs -> SOFTPRESx : register is used for resetting peripherals
    - Eg: DevCfgRegs.SOFTPRES2.bit.EPWM7 = 1u; //*Resetting EPWM7 since we only use PWM1 to 6*
- vSetPeripheralAccess:
    - Setup peripheral access control
        - CPU only
        - No access
    - SysPeriphAcRegs : Register is used
    - Eg:
      SysPeriphAcRegs.ADCA_AC.all = 3u; //CPU access only
      /*



5	4	3	2	1	0
DMA1_ACC		CLA1_ACC		CPU1_ACC	
R/W-3h		R/W-3h		R/W-3h	

Since DMA and CLA access is not required bit 5-2 will be set to 0



CPU1_ACC	R/W	3h	Defines Access control definition for the CPU1 as: 11: Full Access for both read and Write 10: Protected RD Access such that FIFOs, Clear on read registers are not changed + No Write Access 01: Reserved 00: No Read/Write Access to peripheral Reset type: XRSn

For CPU1 read and write access : '11', That is 3.
For No Access: 0

- vLockConfig:
    - For locking SYS configurations, such that one cant change it on run time
    - For Clock:
      ClkCfgRegs.CLKCFGLOCK1.bit.CLKSRCCTL1 = 1u;
    - For CPU:
      CpuSysRegs.CPUSYSLOCK1.bit.PCLKCR2 = 1u;
    - For peripheral access:
      SysPeriphAcRegs.PERIPH_AC_LOCK.bit.LOCK_AC_WR = 1u;
    - Similarly for DMA and CLA sources

Internal Watch Dog

- The watchdog module consists of an 8-bit counter fed by a prescaled clock.
- When the counter reaches its maximum value, the module generates an output pulse 512 WDCLKs wide. This pulse can generate an interrupt or a reset.
- The CPU must periodically write a 0x55 + 0xAA sequence into the watchdog key register to reset the watchdog counter.
- The WDCNTR is reset-enabled when a value of 0x55 is written to the WDKEY. When the next value written to the WDKEY register is 0xAA, then the WDCNTR is reset. Any value written to the WDKEY other than 0x55 or 0xAA causes no action.
- Only a write of 0x55 followed by a write of 0xAA to the WDKEY resets the WDCNTR. No other sequence will trigger reset.
- Sys_vEnableDog():
    - WDCR- Watchdog control register.
    - WdRegs.WDCR.all = 0x0E28u;

      / 0000 – Reserved*

      E – WD CLK PRE DIVIDER = 128

      0 – RESERVED

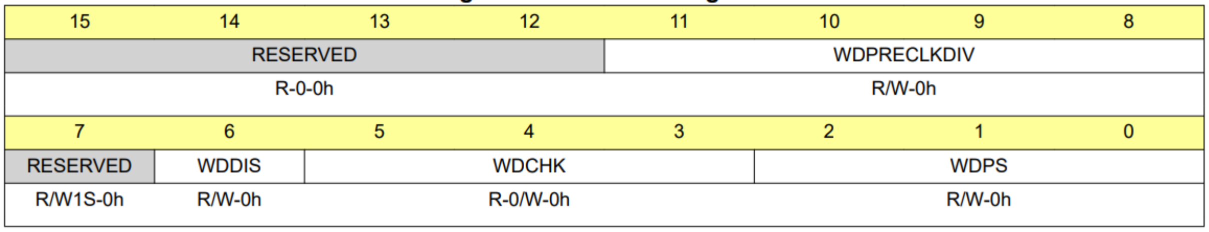
      0 – WDDIS (SINCE WD IS NOT DISABLED)

      101 – WDCHK, watchdog check bits, it should be always 101 else it would cause immediate reset of the core

      *00-   WDPS , Watchdog presacaler =0 */*

      / Enable Watchdog with WDPRECLKDIV = 128*/*

Figure 3-327. WDCR Register



15	14	13	12	11	10	9	8
RESERVED				WDPRECLKDIV			
R-0-0h				R/W-0h			
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESERVED	WDDIS	WDCHK			WDPS		
R/W1S-0h	R/W-0h	R-0/W-0h			R/W-0h		

- Sys_vTriggerDog : Just trigger WD by writing correct sequence
  WdRegs.WDKEY.bit.WDKEY = 0x55u;
  WdRegs.WDKEY.bit.WDKEY = 0xAAu;
- Sys_vTriggerReset: Write incorrect sequence to reset core
  WdRegs.WDCR.all = 0x0E10u;
- vDisableDog:
  temp = WdRegs.WDCR.all & 0x0007u; // store WDPS
  WdRegs.WDCR.all = 0x0068u | temp;

/ 0068 so that,*

1 – WDDIS, Watch dog disabled

101 – WDCHK, watchdog check bits, it should be always 101 else it would cause immediate reset of the core

GPIO.C

Gpio_vInit

```
void Gpio_vInit(void)
        {
          EALLOW;
          /*** Configuration output pin for Window Watchdog- External WD ***/
          GpioCtrlRegs.GPADIR.bit.GPIO28  = 1u; //Configuring as output
          GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 0u; //Setting the pin as GPIO
          GpioDataRegs.GPASET.bit.GPIO28 = 1u; //
        vRead_HwRevision();
        /*Read HwId to determine the Hw revision dependent configuration (DIO, ADC)
        initialisation */
        vInit_Inputs();
        GpioDataRegs.GPACLEAR.bit.GPIO28 = 1;
        /*First trigger of WD. Negative edge triggered*/
        vInit_Outputs();
        #if (DEBUG_PINS_USED)
        vInit_DebugPins();
        #endif /* DEBUG_PINS_USED */
        vLockConfig();
          EDIS;
        }
```

- Main Registers
    - GpioCtrlRegs:
        - GPBDIR: 0 for input, 1 for output
        - GPBMUX2: For muxing, as GPIO or ADC or PWM etc..
    - GpioDataRegs:
        - GPASET = 1;// set the gpio high
        - GPACLEAR = 1//set the gpio low
- vRead_HwRevision: To know hardware revision. Reading some specified gpios will give the HWRevison. 5 GPIOs are confirgured for these 56-59 and GPIO40

```
GpioCtrlRegs.GPBDIR.bit.GPIO56 = 0u;  //Setting it as input pin
GpioCtrlRegs.GPBMUX2.bit.GPIO56 = 0u;  //Muxing to GPIO
```

- vInit_Inputs:
    - Local initialization of GPIO Inputs
    - HSI document provides all the documents related to pins, which pin is configured to what etc..
    - GPIO MUXING : Multiple pins are connected to one single output using multiplexers. The muxing depends on the value of GPxMUX register value.
    - Here GPIODIR will be 0, since input pins
- vInit_Outputs
    - Local init of GPIO outputs
    - MUX and DIR is set here
    - For PWM pins, GPAPUD is also set to one for disabling pull ups in PWM

- vInit_DebugPins();
    - These pins are used for debug purposes
    - These are the same pins used for HWRevision. Once HWRevision is read, we can use these same pins as debug output pins
    - Here GPBDIR = 1, Since it is used as output pins, Eg: To view some signals
- vLockConfig
    - To lock the pins after it has been configured
    - GPALOCK – Locks the pins

      0: Pin configuration lock is unlocked

      1: Pin configuration lock is locked

    - GPACR - Once set, a lock commit can only be cleared by a reset.

      0: Pin configuration lock is unlocked

      1: Pin configuration lock is locked
    - GpioCtrlRegs.GPALOCK.all = 0xFFFFFFFF; //For locking
    - GpioCtrlRegs.GPBCR.all = 0xFFFFFFFF; //For lock commit
    - GPIOs for ePWM and external WatchDog are muxed to input in case of FuSa event, so lock commit may not be set on these pins

vInit_Gpio_PWM

- Here the GPIOs which are used as PWM outputs are muxed to PWM
- Since this muxing was not done, lock commit of these registers was not done in the vLockConfig
- VInitGPIOPWM
    - GpioCtrlRegs.GPALOCK.all = 0u; *// Unlock all registers*
    - GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1u; *//Set muxing to PWM, similarly do for all pins*

  - */* Lock and commit finally registers */*
    GpioCtrlRegs.GPALOCK.all = 0xFFFFFFFF;
    GpioCtrlRegs.GPACR.all = 0xFFFFFFFF;

vSetSafeState

- Setting the GPIOs to safe state in case of any faults
- Mux channel to normal GPIO and set output to low
- Disables PWM for converter by setting those PWM pins low
- vsetSafeState
  - */* Unlock needed registers (without commit) */*
    GpioCtrlRegs.GPALOCK.all = 0u;
  - GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0u; *//Mux to normal GPIO*
  - GpioDataRegs.GPACLEAR.bit.GPIO0 = 1u; *//Clear the PIN*
    Similarly do for all required PINS
  - */* Lock and commit finally registers */*
    GpioCtrlRegs.GPALOCK.all = 0xFFFFFFFF;
    GpioCtrlRegs.GPACR.all = 0xFFFFFFFF;

Dio.C

- For every GPIOs we have a DIO function which can set/clear the GPIO
- It is mainly called by the application layer

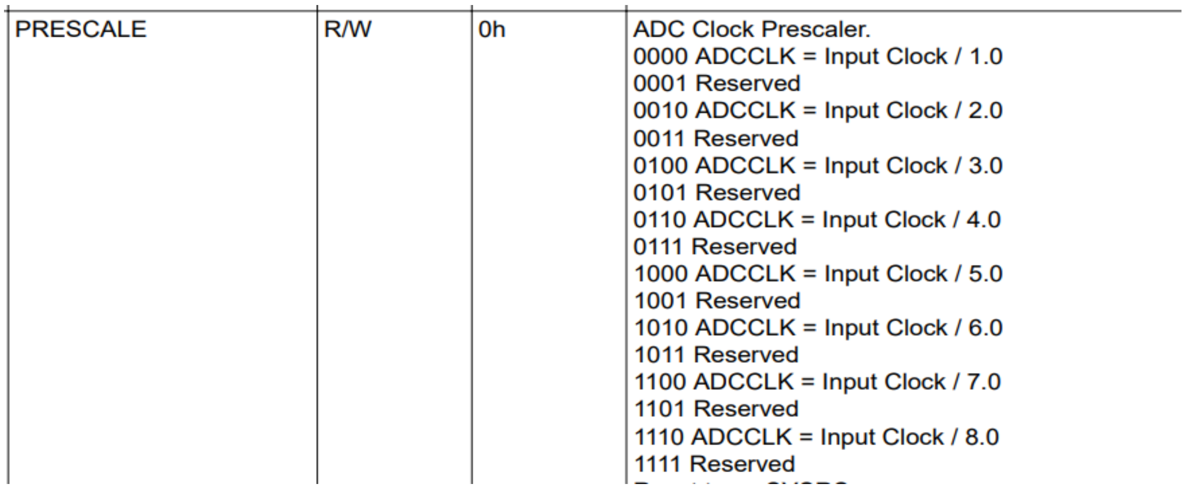
ADC.C

- The ADC module is a 12-bit successive approximation (SAR) style ADC
- Some ADC configurations are individually controlled by the SOCs, while others are globally controlled per ADC module.
- The base ADC clock is provided directly by the system clock (SYSCLK). This clock is used to generate the ADC acquisition window. The register ADCCTL2 has a PRESCALE field that determines the ADCCLK.
- The ADC triggering and conversion sequencing is accomplished through configurable start-of-conversion
- Each SOC is a configuration set defining the single conversion of a single channel.
- Each SOC has its own configuration register, ADCSOCxCTL. In that set there are three configurations:
  - The trigger source that starts the conversion: TRIGSEL
  - The channel to convert: CHSEL
  - The acquisition (sample) window duration : ACQPS

Adc_vInit

- Code:
  vInit_AdcA();
  vInit_AdcB();
  vInit_AdcC();
  /* Init internal 1V2 DCDC - AnalogSubsys component */
  vInit_IntDCDC();
  DELAY_US(ADC_PWRDELAY); /* delay for 1ms to allow ADC time to power up */

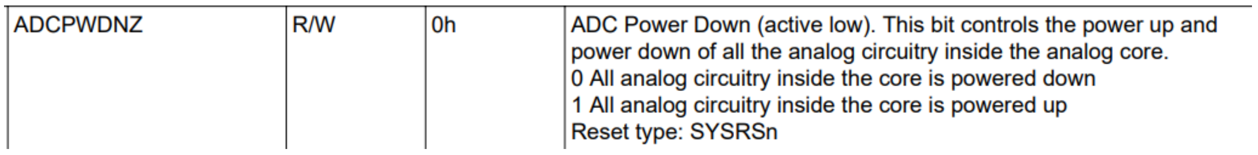
- vInit_AdcA();
  /*Initializes ADC module A*/
  - AdcaRegs.ADCCTL2.bit.PRESCALE = ADC_PRESCALE_2; //2u



PRESCALE	R/W	0h	ADC Clock Prescaler. 0000 ADCCLK = Input Clock / 1.0 0001 Reserved 0010 ADCCLK = Input Clock / 2.0 0011 Reserved 0100 ADCCLK = Input Clock / 3.0 0101 Reserved 0110 ADCCLK = Input Clock / 4.0 0111 Reserved 1000 ADCCLK = Input Clock / 5.0 1001 Reserved 1010 ADCCLK = Input Clock / 6.0 1011 Reserved 1100 ADCCLK = Input Clock / 7.0 1101 Reserved 1110 ADCCLK = Input Clock / 8.0 1111 Reserved

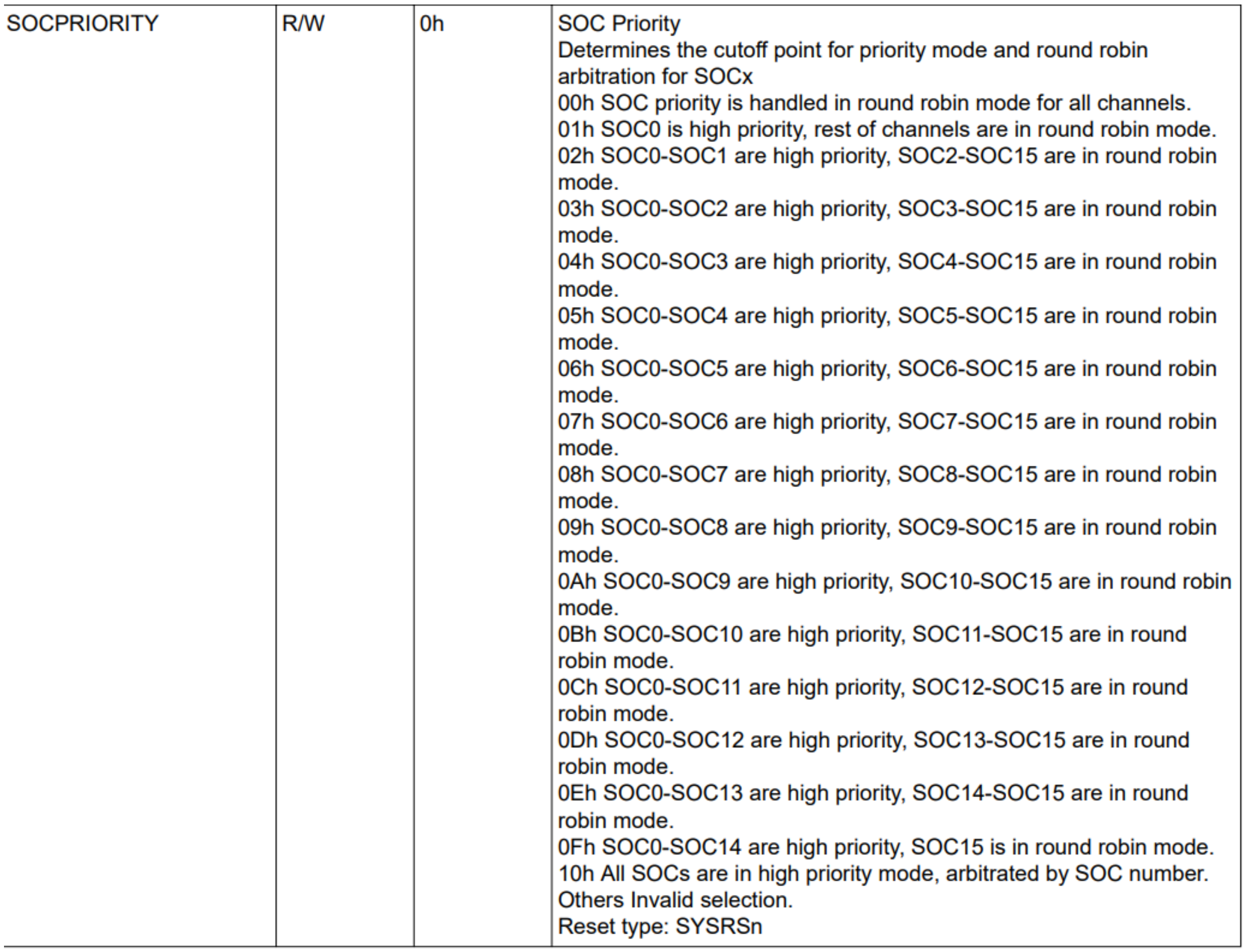
  - AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1u;

    /*Power up the ADC*/



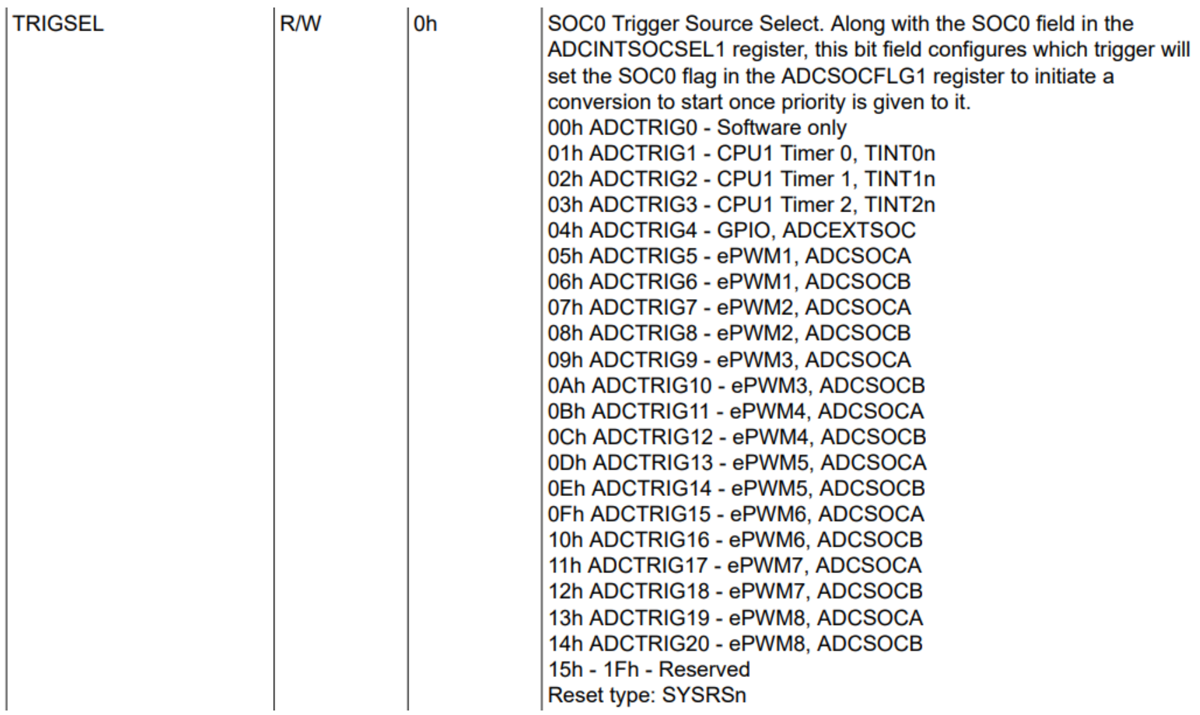
ADCPWDNZ	R/W	0h	ADC Power Down (active low). This bit controls the power up and power down of all the analog circuitry inside the analog core. 0 All analog circuitry inside the core is powered down 1 All analog circuitry inside the core is powered up Reset type: SYSRSn

  - AdcaRegs.ADCSOCPRICTL.bit.SOCPRIORITY = ADC_SOC0_SOC2;
    /*SOC priority control,



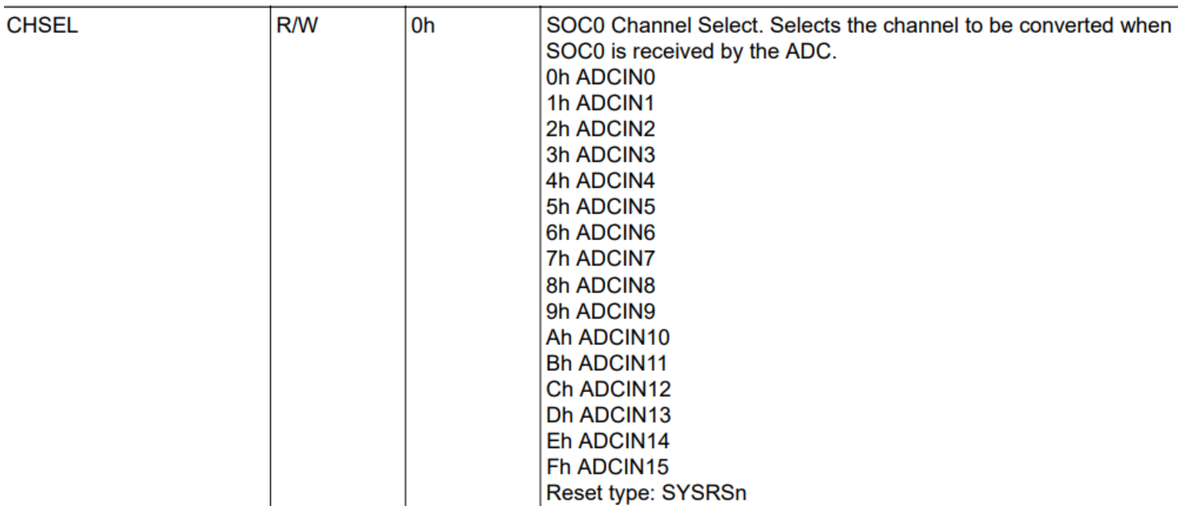
SOCPRIORITY	R/W	0h	SOC Priority Determines the cutoff point for priority mode and round robin arbitration for SOCx 00h SOC priority is handled in round robin mode for all channels. 01h SOC0 is high priority, rest of channels are in round robin mode. 02h SOC0-SOC1 are high priority, SOC2-SOC15 are in round robin mode. 03h SOC0-SOC2 are high priority, SOC3-SOC15 are in round robin mode. 04h SOC0-SOC3 are high priority, SOC4-SOC15 are in round robin mode. 05h SOC0-SOC4 are high priority, SOC5-SOC15 are in round robin mode. 06h SOC0-SOC5 are high priority, SOC6-SOC15 are in round robin mode. 07h SOC0-SOC6 are high priority, SOC7-SOC15 are in round robin mode. 08h SOC0-SOC7 are high priority, SOC8-SOC15 are in round robin mode. 09h SOC0-SOC8 are high priority, SOC9-SOC15 are in round robin mode. 0Ah SOC0-SOC9 are high priority, SOC10-SOC15 are in round robin mode. 0Bh SOC0-SOC10 are high priority, SOC11-SOC15 are in round robin mode. 0Ch SOC0-SOC11 are high priority, SOC12-SOC15 are in round robin mode. 0Dh SOC0-SOC12 are high priority, SOC13-SOC15 are in round robin mode. 0Eh SOC0-SOC13 are high priority, SOC14-SOC15 are in round robin mode. 0Fh SOC0-SOC14 are high priority, SOC15 is in round robin mode. 10h All SOCs are in high priority mode, arbitrated by SOC number. Others Invalid selection. Reset type: SYSRSn

- The above were core configuration
- Now for each SOCs, channel, trigger and acquisition window has to be selected
- Trigger Select,
  Eg: AdcaRegs.ADCSOC7CTL.bit.TRIGSEL = ADC_SOC_TIMER0;



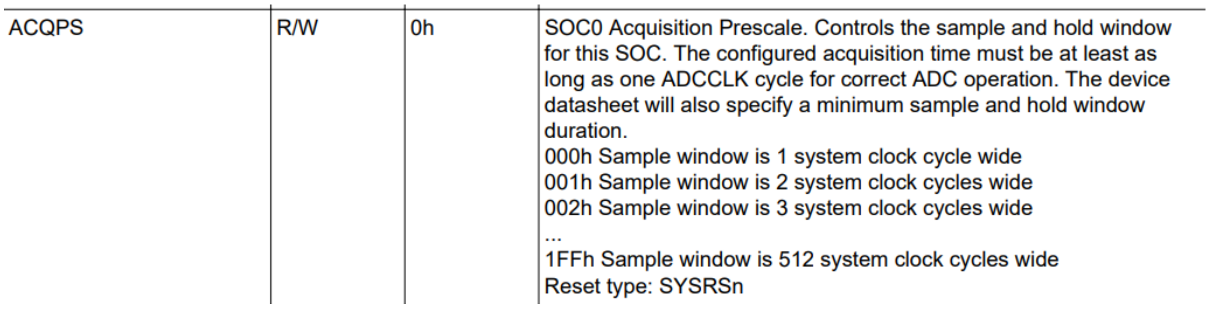
TRIGSEL	R/W	0h	SOC0 Trigger Source Select. Along with the SOC0 field in the ADCINTSOCSEL1 register, this bit field configures which trigger will set the SOC0 flag in the ADCSOCFLG1 register to initiate a conversion to start once priority is given to it. 00h ADCTRIG0 - Software only 01h ADCTRIG1 - CPU1 Timer 0, TINT0n 02h ADCTRIG2 - CPU1 Timer 1, TINT1n 03h ADCTRIG3 - CPU1 Timer 2, TINT2n 04h ADCTRIG4 - GPIO, ADCEXTSOC 05h ADCTRIG5 - ePWM1, ADCSOCA 06h ADCTRIG6 - ePWM1, ADCSOCB 07h ADCTRIG7 - ePWM2, ADCSOCA 08h ADCTRIG8 - ePWM2, ADCSOCB 09h ADCTRIG9 - ePWM3, ADCSOCA 0Ah ADCTRIG10 - ePWM3, ADCSOCB 0Bh ADCTRIG11 - ePWM4, ADCSOCA 0Ch ADCTRIG12 - ePWM4, ADCSOCB 0Dh ADCTRIG13 - ePWM5, ADCSOCA 0Eh ADCTRIG14 - ePWM5, ADCSOCB 0Fh ADCTRIG15 - ePWM6, ADCSOCA 10h ADCTRIG16 - ePWM6, ADCSOCB 11h ADCTRIG17 - ePWM7, ADCSOCA 12h ADCTRIG18 - ePWM7, ADCSOCB 13h ADCTRIG19 - ePWM8, ADCSOCA 14h ADCTRIG20 - ePWM8, ADCSOCB 15h - 1Fh - Reserved Reset type: SYSRSn

- Channel Select,



CHSEL	R/W	0h	SOC0 Channel Select. Selects the channel to be converted when SOC0 is received by the ADC. 0h ADCIN0 1h ADCIN1 2h ADCIN2 3h ADCIN3 4h ADCIN4 5h ADCIN5 6h ADCIN6 7h ADCIN7 8h ADCIN8 9h ADCIN9 Ah ADCIN10 Bh ADCIN11 Ch ADCIN12 Dh ADCIN13 Eh ADCIN14 Fh ADCIN15 Reset type: SYSRSn

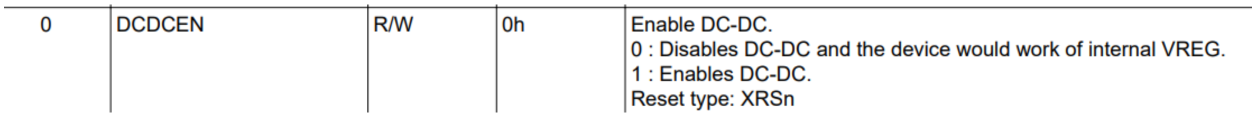
- Acquisition Window Selection,



ACQPS	R/W	0h	SOC0 Acquisition Prescale. Controls the sample and hold window for this SOC. The configured acquisition time must be at least as long as one ADCCLK cycle for correct ADC operation. The device datasheet will also specify a minimum sample and hold window duration. 000h Sample window is 1 system clock cycle wide 001h Sample window is 2 system clock cycles wide 002h Sample window is 3 system clock cycles wide ... 1FFh Sample window is 512 system clock cycles wide Reset type: SYSRSn

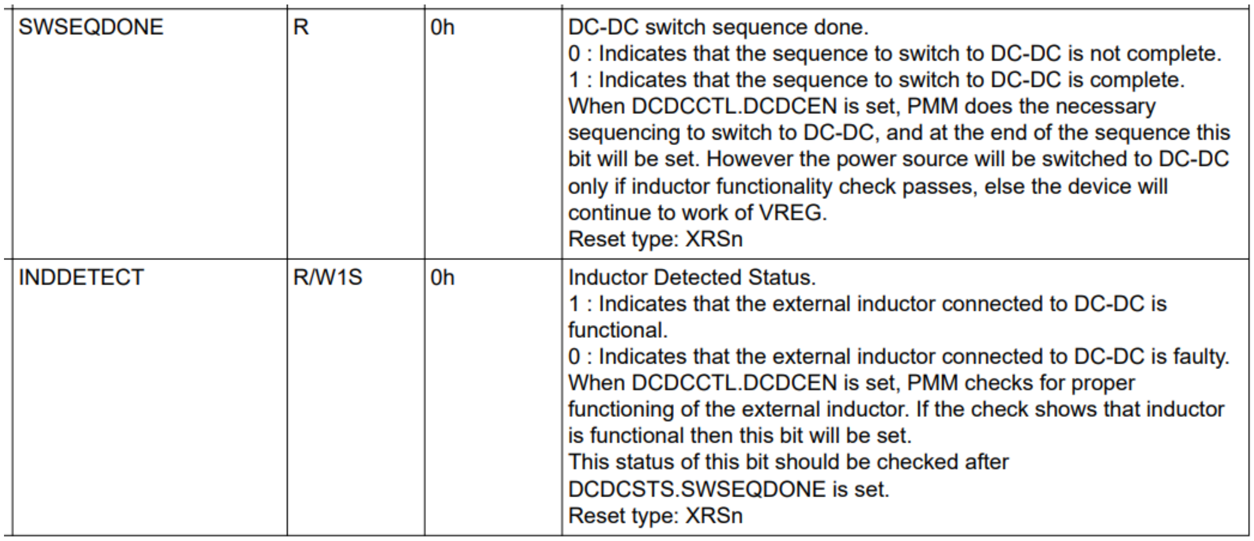
- vInit_IntDCDC()
    - For internal 1.2V DCDC supply
    - Internal cyclic tests or start up tests to check whether internal 1.2V supply is working properly or not
    - Analog subsystem ->DCDCCTL :DC DC control register

```
/* Switch on internal DCDC */
  AnalogSubsysRegs.DCDCCTL.bit.DCDCEN = 1u;
```



0	DCDCEN	R/W	0h	Enable DC-DC. 0 : Disables DC-DC and the device would work of internal VREG. 1 : Enables DC-DC. Reset type: XRSn

```
  DELAY_US(100u);
  /* Check if DCDC is running*/
  if( (AnalogSubsysRegs.DCDCSTS.bit.INDDETECT == 1u) ||
(AnalogSubsysRegs.DCDCSTS.bit.SWSEQDONE == 1u) )
```



SWSEQDONE	R	0h	DC-DC switch sequence done. 0 : Indicates that the sequence to switch to DC-DC is not complete. 1 : Indicates that the sequence to switch to DC-DC is complete. When DCDCCTL.DCDCEN is set, PMM does the necessary sequencing to switch to DC-DC, and at the end of the sequence this bit will be set. However the power source will be switched to DC-DC only if inductor functionality check passes, else the device will continue to work of VREG. Reset type: XRSn
INDDETECT	R/W1S	0h	Inductor Detected Status. 1 : Indicates that the external inductor connected to DC-DC is functional. 0 : Indicates that the external inductor connected to DC-DC is faulty. When DCDCCTL.DCDCEN is set, PMM checks for proper functioning of the external inductor. If the check shows that inductor is functional then this bit will be set. This status of this bit should be checked after DCDCSTS.SWSEQDONE is set. Reset type: XRSn

```
  {     /* Enabling was successful */
    eDcDcOk = OK;
  }

  else
  {
    /* Enabling was not possible, switch off internal DCDC again */
    AnalogSubsysRegs.DCDCCTL.bit.DCDCEN = 0u;
    DELAY_US(100u);
    eDcDcOk = ERR;
```

Adc_vFuSa_ADC2_Init

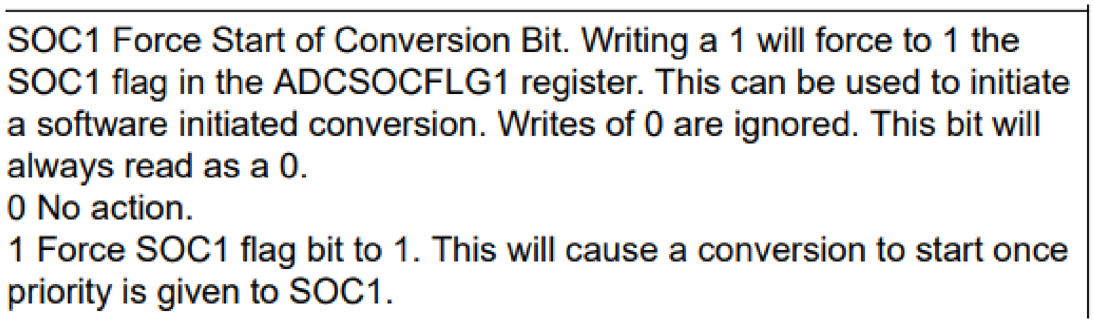
- For ADC Loop back test
- Initializes ADCs for ADC2 loopback check
- We send some signal, then read it back for verification, with the help of ADC and DAC

Adc_vFuSa_ADC2_DeInit

- De-Initialises ADCs after ADC2 loopback check

Adc_Heater_Trigger

- To ADC SOC1 Force heater trigger
- AdccRegs.ADCSOCFRC1.bit.SOC1 = 1u;

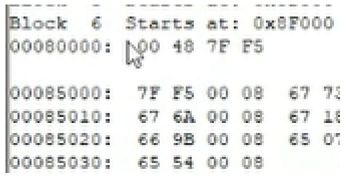

SOC1 Force Start of Conversion Bit. Writing a 1 will force to 1 the SOC1 flag in the ADCSOCFLG1 register. This can be used to initiate a software initiated conversion. Writes of 0 are ignored. This bit will always read as a 0.
0 No action.
1 Force SOC1 flag bit to 1. This will cause a conversion to start once priority is given to SOC1.

Adc_vFuSa_ADC2_Trigger

- Safety Related

Uniflash

- It's a flashing tool, where one can only flash the controller
- No debugging is allowed in this tool
- XDS 110 debugger is used for flashing
- Flash Image - Executable, (.hex/.out)
- In hex file, first address belongs to boot loader. It is a jump instruction.



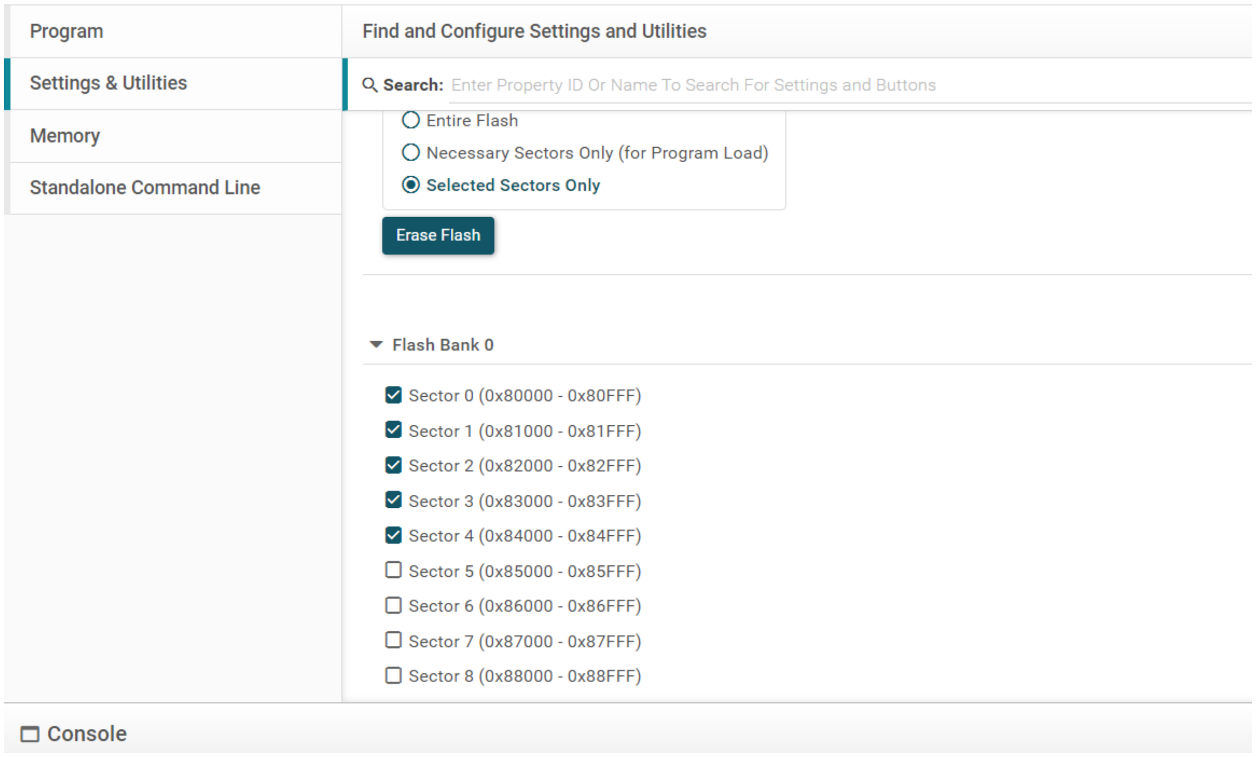
```
Block  6  Starts at: 0x8F000
00080000:  00 48 7F F5

00085000:   7F F5 00 08  67 7
00085010:   67 6A 00 08  67 1
00085020:   66 9B 00 08  65 0
00085030:   65 54 00 08
```

0008000: is the first address and 7F F5 is a jump instruction
Application starts at 8500.

- It is not possible to flash both application and BL at same time.
- So first application is flashed so that BL address is written
- Then deselect the application file, load only BL and signature file and flash it
- Once application is flashed we flash BL and signature files

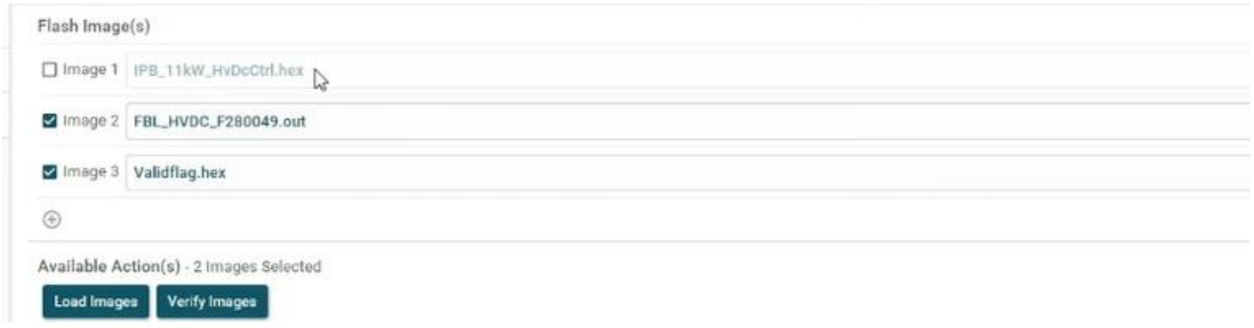
- So when BL is flashed, the memory corresponding application is deselected. Only memory corresponding to BL is rewritten by BL. Ensure that no application memory is erased while flashing BL


Program
Settings & Utilities
Memory
Standalone Command Line
Find and Configure Settings and Utilities
Search: Enter Property ID Or Name To Search For Settings and Buttons
Erase Flash
Flash Bank 0
Console

○ Entire Flash
○ Necessary Sectors Only (for Program Load)
● Selected Sectors Only

- [✓] Sector 0 (0x80000 - 0x80FFF)
- [✓] Sector 1 (0x81000 - 0x81FFF)
- [✓] Sector 2 (0x82000 - 0x82FFF)
- [✓] Sector 3 (0x83000 - 0x83FFF)
- [✓] Sector 4 (0x84000 - 0x84FFF)
- [ ] Sector 5 (0x85000 - 0x85FFF)
- [ ] Sector 6 (0x86000 - 0x86FFF)
- [ ] Sector 7 (0x87000 - 0x87FFF)
- [ ] Sector 8 (0x88000 - 0x88FFF)

- It also contains a signature file
- All the TI tools configuration are saved in the form of .ccxml (code composer xml)



- [ ] Image 1 IPB_11kW_HvDcCtrl.hex
- [✓] Image 2 FBL_HVDC_F280049.out
- [✓] Image 3 Validflag.hex

Available Action(s) - 2 Images Selected

- First flash Image 1: Application (Entire Flash)
- Then flash Image 2 and 3 by selecting appropriate memory : BL and Signature
- For Flashing use Load Images button
- Valid Flag/ Signature: This like a signature BL writes after flashing the application to know that it has verified the boot application. Now after every reset it does not have to verify whether the application is valid or not, it will just check for the signature. If

signature is present then it will know that application was verified at the beginning, and we can jump into application

- When it is flashed using Vflash, one uds service is called which will write this signature. When it is done using debugger, signature has to written manually.

Code Composer Studio

- S/W used for debugging TI C2000 controllers. It is also used as an IDE for programming TI controllers.
- .ccsproject : It gives the information about the project, controller , what type of connection is used
- .project : It gives the name of the project, types of builds, the path of the compiler
- .cproject : these are the configurations that are actually present in the GUI. It contains path to bin file for building process.
- All the above 3 are written in xml format
- The workspace is loaded in .ccxml format (70_Tools ->10_CCS81) for flashing to the controller – Target Configuration
- The .ccxml format in the context of C2000 microcontrollers typically refers to a configuration file used by Code Composer Studio (CCS), which is Texas Instruments' integrated development environment (IDE) for their microcontrollers. It contains device configuration, debug configuration, build and flash configuration
-
  f280049_cla_flash.cmd : Linker command file
- .hex file is generated in such a way that it can boot in stand alone mode. When flashed, the entire contents of the memory is erased (application/BL).
-